



## 2. Licht und Materialien

### Licht und Schatten

In three stehen uns eine Reihe verschiedener Leuchtquellen zur Verfügung. Schatten wirft aber nicht jede Lichtquelle, wir können aus 3 Lichtquellen auswählen, die Schatten werfen:

(DirectionalLight, PointLight, Spotlight)

Andere Lichtquellen, die keinen Schatten werfen, sind als indirekte Lichtquellen gedacht:

(AmbientLight, HemisphereLight).

Wir können entweder mit der zur Verfügung gestellten Datei

**L2-1\_Licht.html**

beginnen (Dort ist ein leerer Rumpf für das Licht und ein leerer Rumpf für die GUI vordefiniert). Alternativ könnt ihr aber auch von der Datei der letzten Übung aus starten (in der die GUI erweitert werden sollte) und dort alle nicht relevanten Einträge in der GUI löschen. Ändert zudem die Bewegung des Kubus wieder auf eine Drehbewegung um die Achsen, so können die Effekte besser unterschieden werden.

Vorbereitung: Deklariert 5 verschiedene Lichtquellen global, so viele wollen wir ausprobieren.

Zum Ein- und Ausschalten der Lichtquellen soll der Hinweis erfolgen, dass alle 3D-Objekte - und damit auch die Lichtquellen - ein Attribut „visible“ haben, das ein Objekt ausblendet, wenn visible auf false steht.

Bei den Lichtquellen starten wir mit dem indirekten Licht, da der Schattenwurf bei den direkten Lichtquellen etwas mehr Erläuterung bedarf.

Dazu vereinbaren wir zunächst die folgenden (in der Funktion initLight() lokale) Variable:

```
let skyColor = 0xe1f2f5; // light blue
let groundColor = 0x6e573c; // brownish orange
let diffusIntensity = 0.4;
let spotIntensity = 3.0;
let spotColor = 0xe6e353; // light yellow
let distance = 35;
```

### AmbientLight

Da ist zunächst das AmbientLight. Damit erstellen wir eine diffuse Lichtquelle, die eine Art Hintergrund- oder Umgebungsbeleuchtung darstellt. AmbientLight wird als allgemeine Lichtquelle für Objekte genutzt. Es kann eher quasi als Tageslicht-Beleuchtung eingesetzt werden. Wir können damit keine Schatten erzeugen.

```
<<constructor>>
AmbientLight( color : Integer, intensity : Float )
```



#### HemisphereLight

HemisphereLight simuliert eine Sonnenbestrahlung, d.h., das Licht kommt direkt von oben über der Szene. Die Lichtintensität verändert sich von der ersten Farbe (oben) zur zweiten Farbe (unten), das Licht erzeugt aber keinen Schatten.

```
<<constructor>>
```

```
HemisphereLight( skyColor : Integer, groundColor : Integer, intensity : Float )
```

#### PointLight

PointLight wird von einem Punkt ausgestrahlt und scheint in alle Richtungen. Objekte, die durch dieses Licht erhellt werden, können Schatten werfen und besitzen einen Lichtradius, bis zu dem das Licht reicht. Neben der Intensität kann eben dieser Lichtradius dem Konstruktor als Parameter übergeben werden. Mit ‚decay‘ kann die Abnahme der Intensität entlang der Lichtstrahlen bestimmt werden. Default ist 1, als physikalisch korrekt wird 2 angegeben.

```
<<constructor>>
```

```
PointLight( color : Integer, intensity : Float, distance : Number, decay : Float )
```

#### SpotLight

Das SpotLight imitiert einen Strahler mit einem kegelförmigen Licht von einem bestimmten Punkt aus in eine bestimmte Richtung. Damit vergrößert sich der Lichtkegel zunehmend von der Quelle. Angestrahlte Objekte werfen einen Schatten. Mit ‚angle‘ kann der Öffnungswinkel des Spots bestimmt werden. Maximum ist  $\pi/2$ . Mit ‚penumbra‘ wird der Anteil des Spotkonus bestimmt, der durch Halbschatten abgeschwächt wird. Default ist 0, Maximum ist 1;

```
<<constructor>>
```

```
SpotLight( color : Integer, intensity : Float, distance : Float, angle : Radians, penumbra : Float, decay : Float )
```

#### DirectionalLight

Das DirectionalLight wirft gleichmäßig vom Ausgangspunkt ausgehende parallele Lichtstrahlen. So, als wenn sie von sehr weit herkommen würden (auch wenn der Ausgangspunkt sehr nah zum Objekt ist). Das DirectionalLight wird ebenfalls Schatten.

```
<<constructor>>
```

```
DirectionalLight( color : Integer, intensity : Float )
```

### Aufgaben 2-1

1. Nehmt die eben angefangene Datei und initialisiert die 5 Lichtobjekte mit je einer Lichtquelle mit den oben angegebenen Werten gemäß den jeweiligen Konstruktoren. Die Werte für ‚angle‘, ‚decay‘ und ‚penumbra‘ können jeweils als Parameterwert übergeben werden.
2. Erstellt die GUI-Implementierung so, dass ein Menüeintrag für die diffusen Lichtquellen vorhanden ist und ein weiterer Menüeintrag für die gerichteten Lichtquellen. Hierfür



eignet sich die JSON-Definition. Beachtet, dass nur zwei diffuse Lichtquellen da sind, aber 3 gerichtete.

## Hilfen

Wir können eine Lichtquelle auch ganz konkret ausrichten (was insbesondere wichtig ist, wenn man ein bestimmtes Element beleuchten möchte).

```
light.target.position.set(0, 0, 0);  
// oder  
light.target.position.set(object);  
scene.add(light.target);
```

Als weitere Hilfestellung bietet sich auch die Möglichkeit, zu kontrollieren, wie die Lichtstrahlen (der Lichtkegel) fallen. Dazu existieren sogenannte Helper, die man immer oder auch nur zu Testzwecken einbauen kann.

```
<<constructors>>  
DirectionalLightHelper( light : DirectionalLight, size : Number, color : Hex )  
PointLightHelper( light : PointLight, sphereSize : Float, color : Hex )  
SpotLightHelper( light : SpotLight, color : Hex )
```

Die Helper-Objekte müssen natürlich auch der Szene hinzugefügt werden.

## Zusatzaufgabe

Probiert aus, was die Parameter ‚size‘, und ‚sphereSize‘ bewirken. Beschreibt die Wirkung mit eigenen Worten.

## Schattenwurf

Wie wirft nun aber das Licht eigentlich Schatten? Zunächst einmal muss das Objekt, das einen Schatten werfen soll, dafür auch geeignet sein! Was nicht grundsätzlich so ist. Geeigneten Materialien sind:

(MeshPhongMaterial, MeshPhysicalMaterial, MeshStandardMaterial, MeshToonMaterial)

Das heißt, dass z.B. MeshBasicMaterial nicht geeignet ist, Schatten entstehen zu lassen.

Daneben müssen wir aber die einzelnen Dinge noch ertüchtigen, damit sie Schatten erzeugen (auch Schatten werden ja vom Renderer errechnet).

Als Erstes muss der Renderer konfiguriert werden, damit er überhaupt Schattenwurf berechnen kann.

```
renderer.shadowMap.enabled = true; // Schattenerzeugung einschalten
```

Dann muss das richtige Material ausgewählt werden (siehe oben) und danach aber auch ertüchtigt werden, damit es Schatten wirft.



## 3D-Programmierung

### Licht und Materialien

Es sind bei jedem Mesh zwei Eigenschaften, die aktiviert (oder deaktiviert) werden können/müssen.

Ein Mesh kann Schatten empfangen (also Schatten anzeigen):

```
mesh.receiveShadow = true;    // Schatten empfangen können
```

Das Mesh kann natürlich auch Schatten erzeugen:

```
mesh.castShadow = true;       // Schatten erzeugen können
```

Und das Licht muss auch Schatten erzeugen. Dazu muss auch noch ein Schattenbereich definiert werden und das Licht erzeugt den Schatten mit einem der Kamera vergleichbaren Mechanismus.

```
light.castShadow = true;      // Schatten erzeugen können  
// Schattenkarte Breite Höhe  
light.shadow.mapSize.width = 512;  
light.shadow.mapSize.height = 512;  
// Schattenkamera Nahbereich und Fernbereich  
light.shadow.camera.near = 5;  
light.shadow.camera.far = 100;
```

Three verwendet Shadow Mapping. Dabei wird getestet, ob ein Pixel von einer Lichtquelle aus sichtbar ist. Die Shadow Map ist eine aus Sicht der Lichtquelle erzeugte Tiefenkarte, die Informationen über den Abstand von Objekten zur Lichtquelle enthält. Mit Hilfe der Tiefenkarte wird bestimmt, ob ein Pixel hinter einem anderen Pixel aus Sicht der Lichtquelle ist. Wenn ja, dann wird es abgedunkelt dargestellt (es liegt im Schatten). Das Verfahren ist etwas weniger genau als andere Verfahren dafür verhältnismäßig performant umsetzbar.

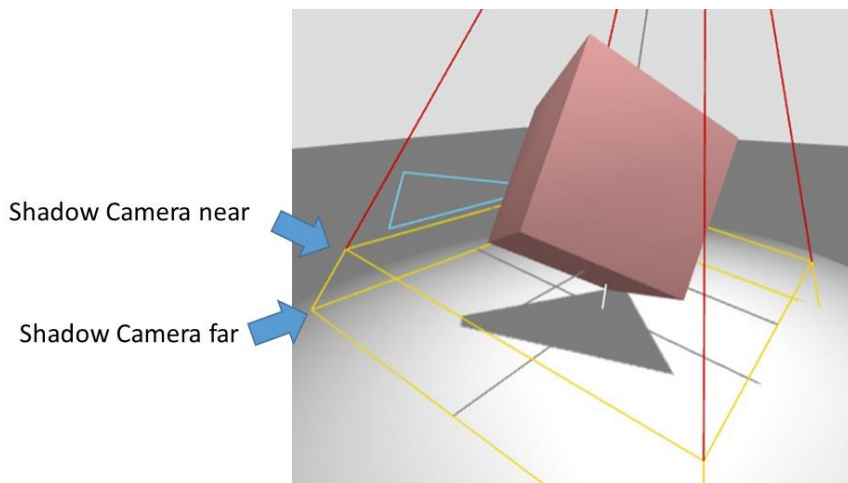
Je größer die mapSize definiert wird, umso größer der Rechenaufwand. Standardeinstellung ist 512x512. Die Angaben müssen ein Vielfaches von 2 sein.

Die camera bildet die Sicht der Lichtquelle auf die Szenerie ab. Wie bei der Kamera, die uns die Szene zeigt (unsere Sicht) gibt es einen Nahbereich und einen Fernbereich, über die hinaus kein Schatten rendert wird. Auch diese Kamera können wir mit einem Helper anzeigen lassen (siehe Abbildung unten).

```
scene.add(new THREE.CameraHelper(light.shadow.camera));
```

Der gezeigte Strahlengang ist bewusst so angelegt, dass man sehen kann, dass immer nur der Teil des Kubus, der zwischen Nah- und Fernbereich liegt, Schatten erzeugt. Man sieht deutlich, dass nur der untere Abschnitt des Kubus einen Schatten erzeugt.

(Hinweis: die far-Einstellung scheint beim SpotLight





nicht zu wirken. Dazu bedarf es des `DirectionalLight`. Dargestellt ist hier jedoch die Szene mit dem `SpotLight`, wo die `near`-Einstellung den gezeigten Effekt hervorruft).

Um einen sichtbaren optischen Effekt zu erzeugen, fügen wir noch eine Fläche unserer Szene hinzu und drehen sie um  $90^\circ$  um die x-Achse, damit wir quasi eine Fläche unter unserem Kubus haben.

### Aufgabe A2-2

1. Erzeugen Sie ein Mesh mit einer `PlaneBufferGeometry` der Größe 200 mal 200 und mit einer Oberfläche aus `MeshToonMaterial({ color: 0xffffff, side: THREE.DoubleSide })` und drehen das Mesh um  $90^\circ$  um die x-Achse und verschieben Sie es soweit auf der y-Achse, bis der Kubus nicht mehr in die Ebene eintaucht. Fügen Sie das Mesh der Szene hinzu.
2. Ertüchtigen Sie den Renderer, das Licht und alle Objekte wie oben beschrieben.

### Materialien

Kommen wir zu den Materialien, die wir bereits verwendet haben, ohne uns näher damit zu befassen. Deswegen betrachten wir mal alle diejenigen Materialien, die Schattenwurf anzeigen. Speichert dazu die eben noch bearbeitete Datei (`L2-A2_Licht.html`) unter dem Namen

`L2-3_Material.html`

ab.

### Aufgabe (Vorbereitung des Szenarios)

1. Damit der Effekt der einzelnen Materialien besser zur Geltung kommt, baut eine kleine Kugel (`SphereGeometrie(2, 32, 32)`) zwischen Lichtquelle und dem bisherigen Würfel in die Szene ein (siehe Bild unten). Der erste Parameter ist der Radius der Kugel, die beiden weiteren Parameter sind die Anzahl Segmente in Breite und Höhe. Minimum 3, wird die Kugel umso runder, je mehr Segmente eingesetzt werden.
2. Löscht in der neuen Datei die bisherigen Menüeinstellungen der GUI, wir arbeiten mit einer festen Lichteinstellung und variieren diesmal das Material. Verwendet das `AmbientLight` mit einer Intensität von 0,2 und ein `SpotLight` mit der Intensität 3.



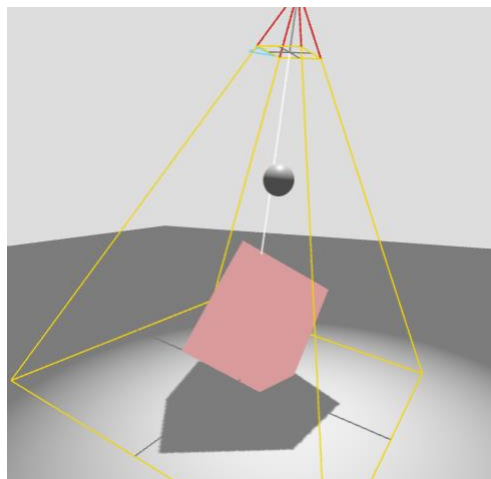
Nun modifizieren wir das Material unseres großen Würfels.

Starten wir mit dem MeshBasicMaterial.

```
matBasic = new THREE.MeshBasicMaterial({  
  color: 0xdd5555,  
});
```

Wir können dem Material Farbe mitgeben (siehe oben). Daneben können wir ihm (wie jedem anderen Material aber auch) Transparenz mitgeben.

```
transparent: true, opacity: 0.5,
```

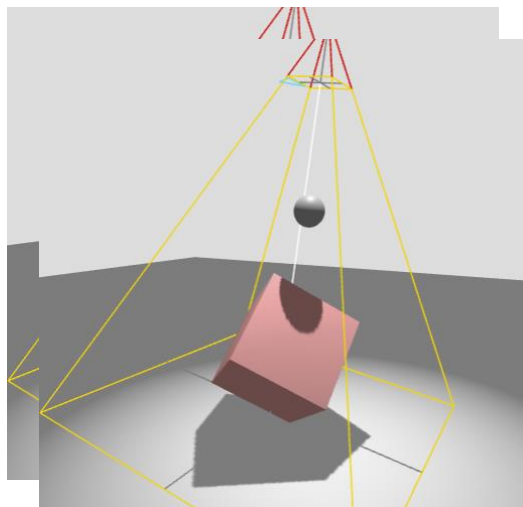


Alle Parameter sind im JSON-Style, Komma-separiert anzugeben. Wie alle Materialien kann man die Parameter gleich beim Erstellen mitgeben (wie es im Beispielcode auch gemacht wurde) oder wir können es nachträglich über die Attribute des Objekts definieren oder ändern. Wie man sehen kann, erzeugt das MeshBasicMaterial Schatten, kann aber selbst keins anzeigen.

```
material.color.set(0x55ff55);
```

Dann kommt das MeshLambertMaterial, das im Gegensatz zum MeshBasicMaterial eine Reflektivität aufweist und damit bereits geeignet sein kann, einfache (reale) Materialien nachzuahmen. Damit haben wir aber bereits (siehe oben) ein Material, auf dem Schattenwurf dargestellt werden kann.

```
matLambert = new THREE.MeshLambertMaterial({  
  color: 0xdd5555,  
});
```



Mehr können wir dann mit dem MeshPhongMaterial anfangen, dem wir auch so etwas wie Glanz mitgeben können. Daneben können wir über das Attribut specular auch die Reflektionsfarbe modellieren. Und dann haben wir noch das Attribut shininess, mit dem wir die Helligkeit der Reflektion bestimmen können.

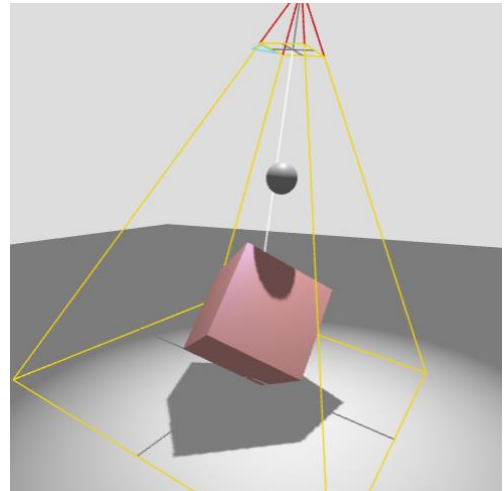


## 3D-Programmierung

### Licht und Materialien

Definieren wir wieder unser Material für den Kubus und geben neben der Helligkeit der Reflektion noch eine Farbe an, die bei der Reflektion mit der Kubus-Farbe vermischt wird (wie eine Lichtbrechung an der Oberfläche), dann erhalten wir das nebenstehende Ergebnis.

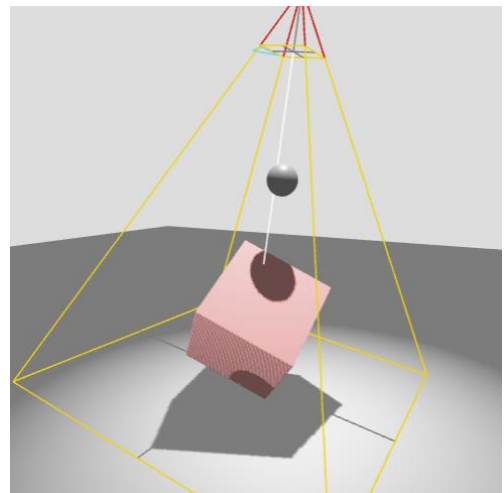
```
matPhong = new THREE.MeshPhongMaterial({  
    color: 0xdd5555,  
    specular: 0x5555ff,  
    shininess: 30,  
});
```



Man erkennt deutlich an der Kubuskante das reflektierte Licht, etwas undeutlicher der ins Bläuliche gehende Ton am Randbereich der Reflektion.

Dann haben wir noch das MeshToonMaterial, das dem MeshPhongMaterial recht ähnlich ist, mit Ausnahme der Art und Weise, wie der Schattenwurf dargestellt wird.

```
matToon = new THREE.MeshToonMaterial({  
    color: 0xdd5555,  
});
```



Man könnte die Darstellung auch als übertrieben plakativ darstellen. Man kann erkennen, dass die verschiedenen Intensitäten, die beim MeshPhongMaterial noch zwischen Ober- und Unterseite erkennbar sind, zwei Farben gewichen sind: hell oder dunkel.

Diese einfachen Materialien haben den Vorteil, dass sie durch den Rechner rasch berechenbar sind. Je realistischer der Oberflächeneindruck werden soll, desto größer wird auch der Rechenaufwand.

Ein weiter verbesserter optischer Eindruck kann durch die sogenannten physikalischen Materialien erzielt werden.

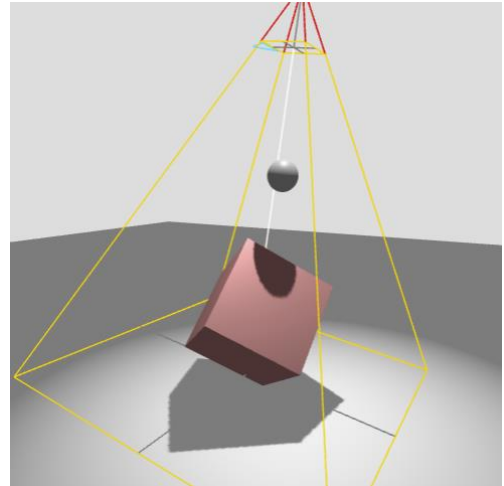
Da steht uns zunächst das MeshStandardMaterial zur Verfügung. Die Reflektion, die beim MeshPhongMaterial mit der shininess angegeben wird, benötigt hier zwei Angaben: die roughness und die metalness. Beide Angaben müssen im Bereich von 0 bis 1 liegen. Je rauer die





Oberfläche ist, desto geringer ist ihre Fähigkeit, Licht zu reflektieren. Durch die Kombination dieser beiden Parameter kann ein schöner metallischer Glanz erzielt werden.

```
matStandard = new THREE.MeshStandardMaterial({  
  color: 0xdd5555,  
  specular: 0x0055ff,  
  roughness: 0.5,  
  metalness: 0.7,  
});
```

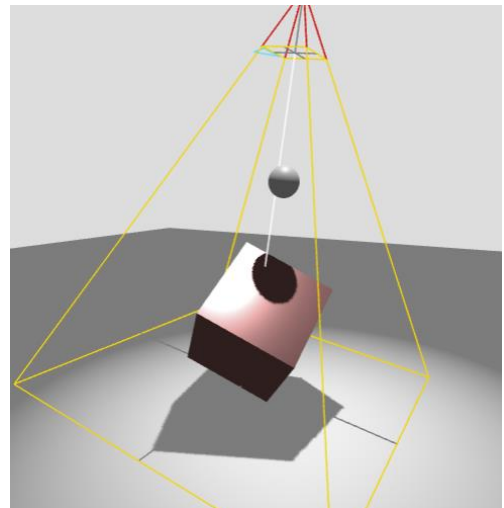


Das nächste physikalische Material, das MeshPhysicalMaterial, fügt zwei weitere Parameter hinzu: clearcoat und clearcoatRoughness.

Mit beiden Parametern lässt sich der Glanzbereich weiter verfeinern.

Beide Parameter gehen wieder von 0 bis 1. Clearcoat variiert die spiegelnde Fläche, die Rauigkeit kann für diesen Bereich separat eingestellt werden.

```
matPhysic = new THREE.MeshPhysicalMaterial ({  
  color: 0xdd5555,  
  specular: 0x0055ff,  
  roughness: 0.5,  
  metalness: 0.7,  
  clearcoat: 0.5,  
  clearcoatRoughness: 0.7,  
});
```



Man kann im nebenstehenden Beispiel die Veränderung in der spiegelnden Oberfläche sehen. Wir haben hier weitere Abstufungen, mit denen es uns möglich sein soll, den Eindruck eines Körpers realistischer zu gestalten.

Weitergehende Möglichkeiten werden wir bei den maps kennen lernen.

## Aufgabe 2-3

1. Implementieren Sie alle Materialien und ändern Sie die GUI so ab, dass die Materialien mit dem Umschalten wechseln. Dem Cube-Mesh können die ein Material einfach durch `cube.material = material` zuweisen. Mit dem nächsten Renderer-Aufruf wird das dann übernommen.

## Zusatzaufgabe Z2

2. In den Unterlagen finden Sie ein Video, das einen Kubus in kreisförmiger Bewegung und zusätzlich Auf- und Ab zeigt. Versuchen Sie, das Szenario nachzugestalten.